Chapter 6

Chapter 6 The Usage of LPM Macro Module

IP

- Intel PSG
  - MegaCore
  - LPM

- The macro modules and LPM functions of Quartus II include：
  - Arithmetic component: adder, multiplier, accumulator, etc.
  - Combinational circuit: multiplexer, comparer, LPM gate function, etc.
  - I/O component: PLL, etc.
  - Memory: FIFO、RAM、ROM, etc.
  - IP of single-chip computer, etc.

The Example of Invoking Macro Module of Counter

- This section gives the general usage methods of MegaWizard Plug-In Manager for the same type of macro modules by introducing the process of invoking and testing the LPM counter LPM_COUNTER.

The Invoking of the Text Code of the Counter LPM Module

Open the MegaWizard Plug-In Manager

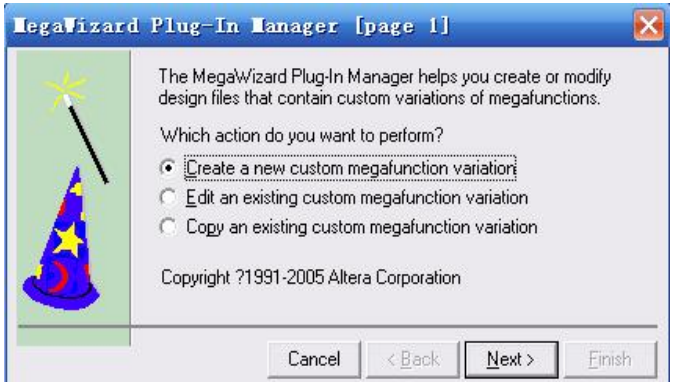


Figure: Customizing new macro function block

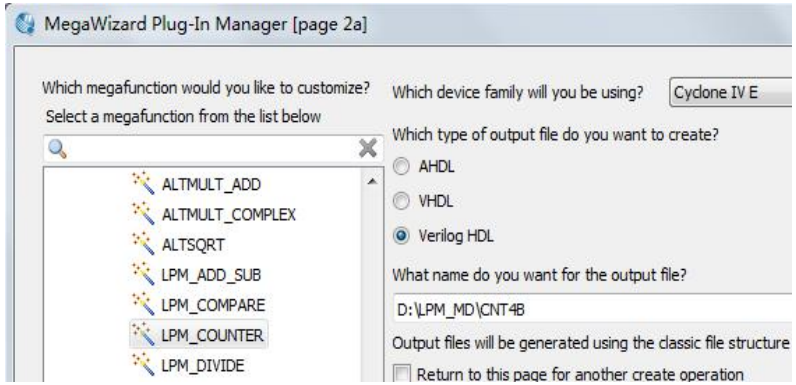


Figure: Setting the LPM macro function block

Set up the project folder，for example, d:\LPM_MD, and choose Tools->MegaWizard Plug-In Manager。

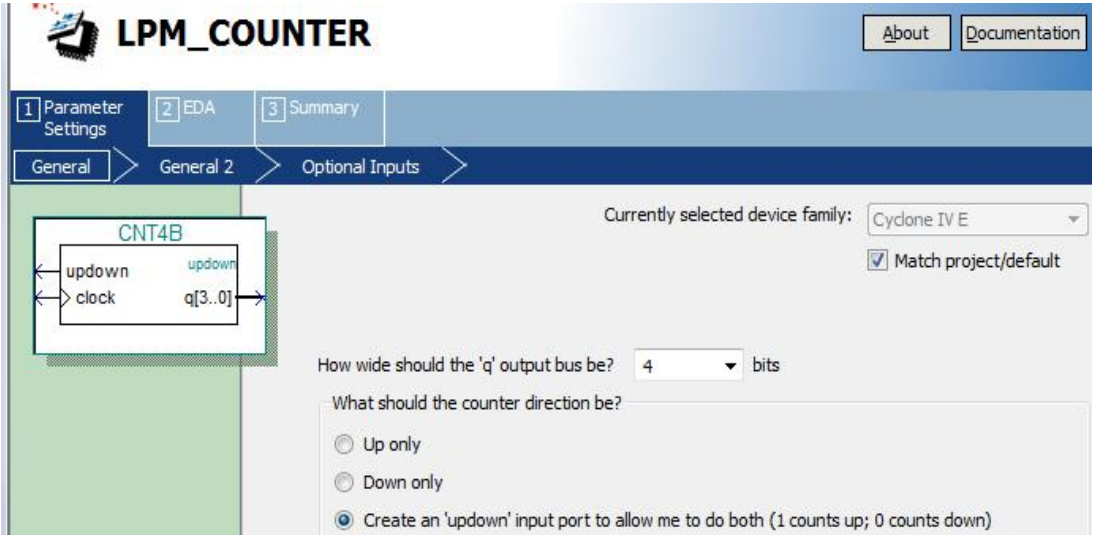


Figure: The setting of 4-bit counter with addition and subtraction

Choose 4-bit counter and

choose "Create an updown input…", which makes the counter have the control functionality of add/subtract.

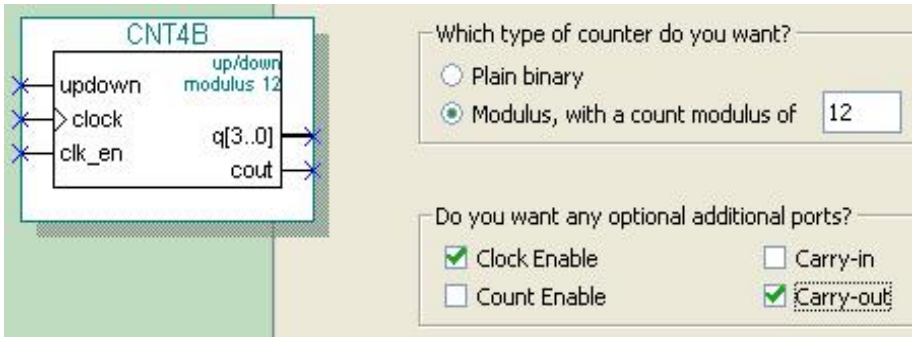


Figure: Setting the counter with the clock enable and carry output

Plain binary: common binary counter

Modulus…: counter with the modulus of …

Clock Enable: clock enable control

Carry-out: carry-out

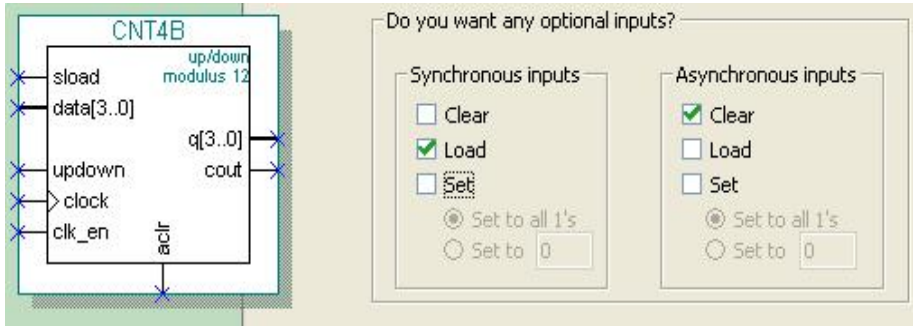


Figure: Adding 4-bit parallel data preset functions

Choose synchronous load and asynchronous clear

The above-mentioned processes generate the Verilog file of the LPM counter, named "CNT4B.v", which can be invoked by the higher level Verilog program as the counter component.

6.1.2 Application of LPM Counter Code and Parameter Transmission Statement

- **[Example]**

```
module CNT4B (aclr, clk_en, clock, data, sload, updown, cout, q);
    input aclr, clk_en;                    //asynchronous clear,1 clear; clock enable, 1 enable, 0 disable
    input clock, sload;                    //clock input; synchronous preset load control, 1 load, 0 count
    input [3:0] data; input   updown;      //4-bit preset number, and addition and subtraction control, 1 addition, 0 subtraction
    output  cout;   output [3:0]  q;       //carry output and 4-bit count output
    wire  sub_wire0;   wire [3:0] sub_wire1;                    // Defining internal connections
    wire  cout = sub_wire0;                // The same assignment statement as assign
    wire [3:0] q = sub_wire1[3:0];         // The same assignment statement as assign
lpm_counter  lpm_counter_component( // Note that the unused ports in the instantiated statement must be connected to the specified level.
    .sload(sload), .clk_en(clk_en), .aclr(aclr),
    .data(data), .clock(clock), .updown(updown),
    .cout(sub_wire0), .q(sub_wire1), .aload(1'b0),
    .aset(1'b0), .cin(1'b1), .cnt_en(1'b1),
    .eq(),  .sclr(1'b0), .sset(1'b0));
defparam
    lpm_counter_component.lpm_direction = "UNUSED",          // Unused unidirectional counting parameters
    lpm_counter_component.lpm_modulus = 12,                  //counter with modulus of 12
    lpm_counter_component.lpm_port_updown = "PORT_USED", // Use the addition and subtraction count
    lpm_counter_component.lpm_type = "LPM_COUNTER",          // Counter type
    lpm_counter_component.lpm_width = 4;                     // Counting bit width
endmodule
```

- The general description of the parameter transmission statement *defparam* is given as follows:
- `defparam < macro module component instantiation name >. < macro module parameter name > = < parameter value >`

[Example]

```
module REG24B (d, clk, q);
    input [23:0] d;     input  clk;
    output [23:0] q;
    lpm_ff  U1(.q (q[11:0]), .data (d[11:0]), .clock (clk));
       defparam U1.lpm_width = 12;
    lpm_ff  U2(.q(q[23:12]), .data(d[23:12]), .clock(clk));
       defparam U2.lpm_width = 12;
endmodule
```

For invoking the counter file "CNT4B.v" , testing and implementing the counter, a program must be designed to instantiate it. Example of 6-3 is used to realize the functionality.

[Example]

```
module CNT4BIT (RST,ENA,CLK,DIN,SLD,UD,COUT,DOUT);
    input RST, ENA, CLK, SLD,UD ;
    input [3:0]  DIN;
    output  COUT;    output [3:0]  DOUT ;
    CNT4B  U1(.sload (SLD), .clk_en (ENA), .aclr (RST), .cout (COUT),
            .clock (CLK), .data (DIN),  .updown (UD), .q (DOUT));
endmodule
```

6.1.3 Project Creation and Simulation Testing

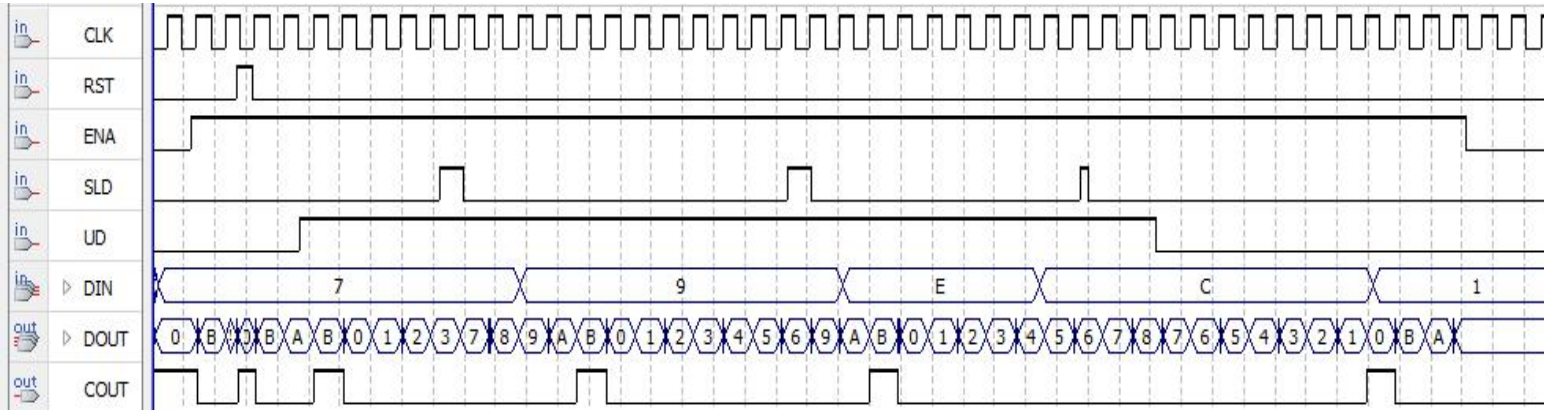


Figure: The simulation waveform of CNT4BIT.v

6.2 Example of Building Attribute Control Multiplier

For implementing the multiplier, if use conventional method, the synthesized multiplier will occupy large logic resource and the speed may not be high. The useful method is to invoke the embedded hardware multiplier in the FPGA and this type of multiplier is commonly used in DSP technology. Thus, this type of multiplier is called DSP module.

`/* synthesis multstyle = "logic" */`	The way of pure combinational logic
`/* synthesis multstyle = "dsp" */`	The way of invoking FPGA embedded multiplier

[Example]

```
module MULT8 (A1,B1,A2,B2,R1,R2) ;
    output signed[15:0] R1, R2 ;      // Defining the output of the signed data
                                      //type
    input signed[7:0] A1,B1,A2,B2;    // Defining the input of the signed data
                                      //type
   wire [15:0] R2, R1 ;
   assign R1 = A1 * B1 ;     assign R2 = A2 * B2 ;
endmodule
```

The multiplier which uses R2 as the output port is constructed by the macro unit utilizing the way of pure combinational logic.

The multiplier which uses R1 as the output port is constructed by invoking the embedded multiplier in FPGA.

- `wire [15:0] R2 /* synthesis multstyle = "logic" */`

- `wire [15:0] R2, R1 /* synthesis multstyle = " logic " */`

If the multiplier in the overall module is required to be constructed by using DSP module, the program can be written as follows:

- `module andd(A1,B1,A2,B2,R1,R2) /* synthesis multstyle = "dsp" */;`

Flow Status	Successful - Tue May 23 21:32:59 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	190 / 55,856 (< 1 %)
Total combinational functions	190 / 55,856 (< 1 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 308 (0 %)
Total PLLs	0 / 4 (0 %)

Figure: The compilation report of completely using logic macro units to construct the multiplier

Flow Status	Successful - Tue May 23 21:30:42 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	0 / 55,856 (0 %)
Total combinational functions	0 / 55,856 (0 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	2 / 308 (< 1 %)
Total PLLs	0 / 4 (0 %)

Figure: The compilation report of invoking DSP module

The compiling report of example 6-4

Flow Status	Successful - Tue May 23 21:32:59 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	190 / 55,856 (< 1 %)
Total combinational functions	190 / 55,856 (< 1 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 308 (0 %)
Total PLLs	0 / 4 (0 %)

If the multiplier in the overall module is required to be constructed by using DSP module, the program can be written as follows:

```
module andd(A1,B1,A2,B2,R1,R2)  /* synthesis multstyle = "dsp" */;
```

Flow Status	Successful - Tue May 23 21:30:42 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	0 / 55,856 (0 %)
Total combinational functions	0 / 55,856 (0 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	2 / 308 (< 1 %)
Total PLLs	0 / 4 (0 %)

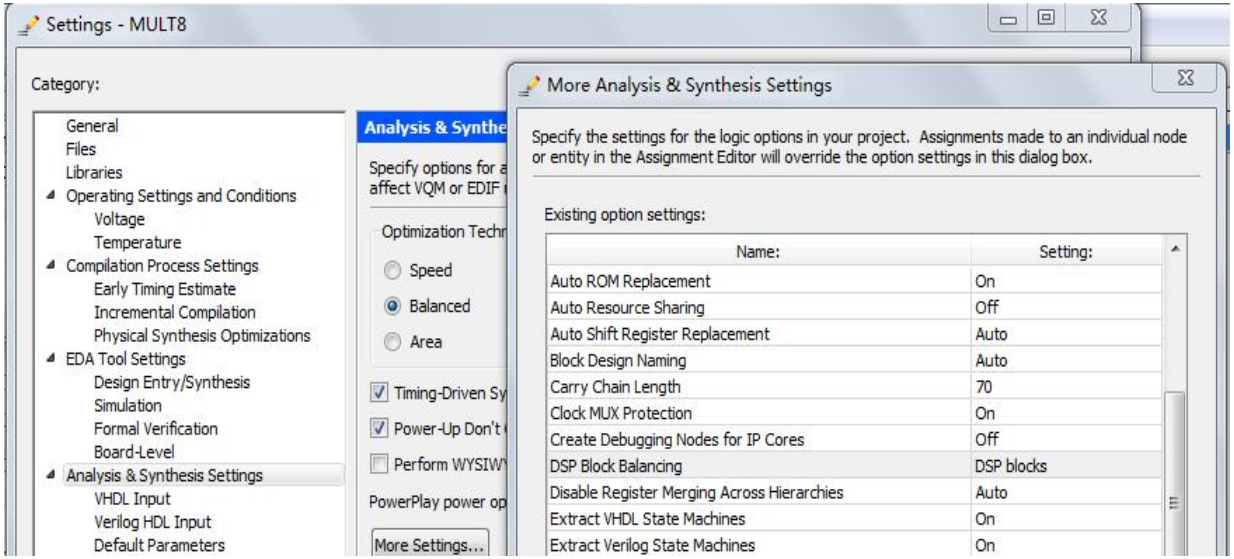


Figure: Selecting DSP Block Balancing as DSP blocks

6.3 Usage of Macro Block of LPM_RAM

- In the design and development of involving memory applications such as RAM and ROM, invoking LPM module-type memory is the most convenient, most economical, most effective, and most efficient way to satisfy the design requirements. The following introduces the related technologies of using Quartus II to invoke LPM_RAM, including simulation test, generation of initialization configuration file, instantiation program expression, related attribute application and Verilog language description of memory.

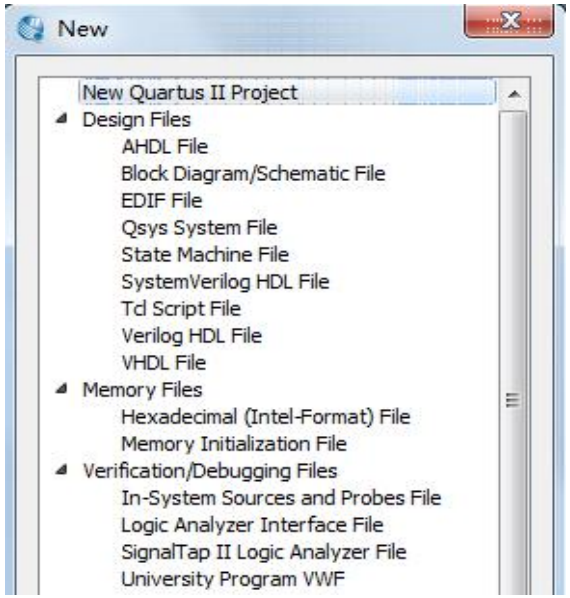
6.3.1 Initialization File and Its Generation

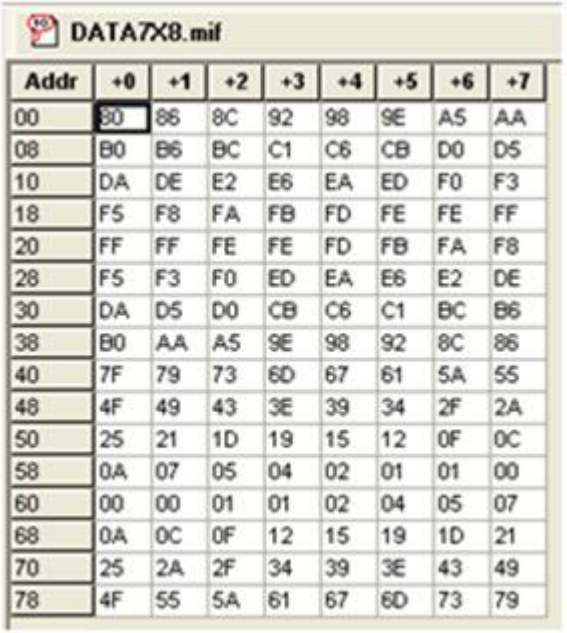
- In the design and development of the RAM and ROM applications, invoking LPM memory is the most convenient and cost-effective way for satisfying the design requirements.
- The initialization file of the memory is the data or code that can be configured in RAM or ROM. In the EDA design, the memory code file set or designed by the EDA tools is automatically invoked in the unified compilation.
- Quartus II can accept two types of initialization files: .mif and .hex.

⌘ 1. .mif format file

（1）The method of direct editing

File -> New -> Memory Initialization File





DATA7X8.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	80	86	8C	92	98	9E	A5	AA
08	B0	B6	BC	C1	C6	CB	D0	D5
10	DA	DE	E2	E6	EA	ED	F0	F3
18	F5	F8	FA	FB	FD	FE	FE	FF
20	FF	FF	FE	FE	FD	FB	FA	F8
28	F5	F3	F0	ED	EA	E6	E2	DE
30	DA	D5	D0	CB	C6	C1	BC	B6
38	B0	AA	A5	9E	98	92	8C	86
40	7F	79	73	6D	67	61	5A	55
48	4F	49	43	3E	39	34	2F	2A
50	25	21	1D	19	15	12	0F	0C
58	0A	07	05	04	02	01	01	00
60	00	00	01	01	02	04	05	07
68	0A	0C	0F	12	15	19	1D	21
70	25	2A	2F	34	39	3E	43	49
78	4F	55	5A	61	67	6D	73	79

1. .mif format

（2）The method of file editing

Example:

```
DEPTH=128;          The number of data in memory
WIDTH=8;            Width of output data
ADDRESS_RADIX = HEX;  The data type of address
DATA_RADIX = HEX;     The data type of memory
CONTENT
BEGIN
0000      :     0080;
0001      :     0086;
0002      :     008C;
   ...
007E      :     0073;
007F      :     0079;
END;
```

Regular editor can be used to design MIF file. Address and data are both hexadecimal.

Save as .mif file

1. .mif format

（3）specific mif file generator

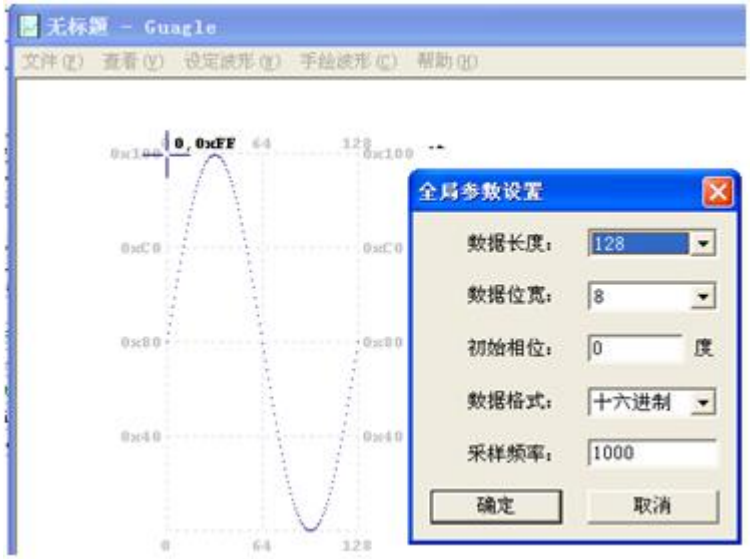


Figure: Generation of .mif sinusoidal waveform file by using mif generator

Figure: Open .mif file

2. .hex format file

（1）method 1：New -> Hexadecimal (Intel-Format) File ->save as .hex format file

（2）method 2：The data is edited in HEX data editing window by using assembly program editor and .hex format file is generated by using assembly compiler.

3. .dat format file

.mif and .hex format file is related with the specific development software, as the invoking of them in Verilog text is necessily required to use the stipulated property expression of Quartus II.

However, the invoking of .dat format data file can be realized directly by using standard Verilog statements. The data format of .dat file is simplest and its form is given as follows：

```
00   E5   6D ... 34
```

6.3.2 Invoking LPM_RAM by Schematic Diagram Method

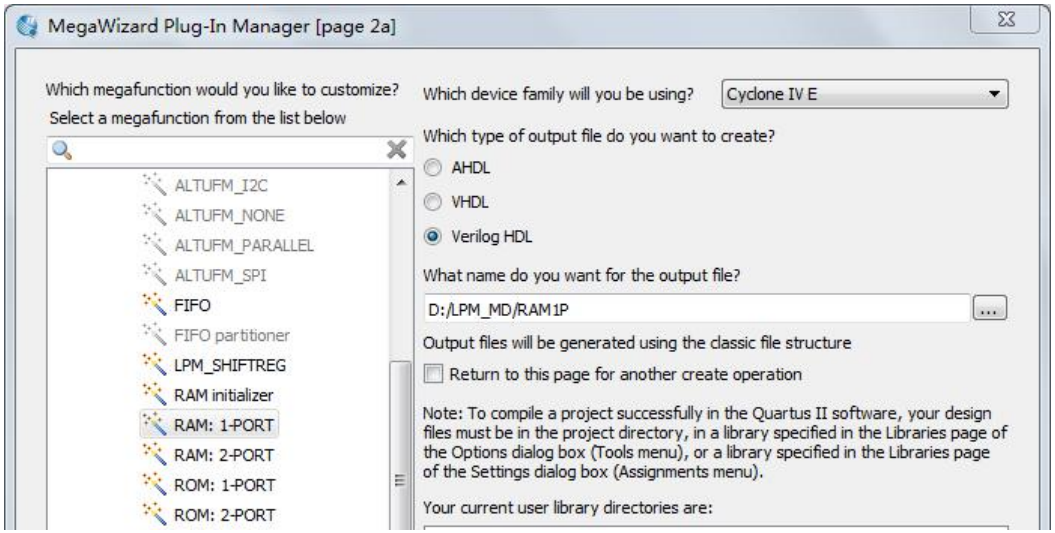


Figure: Invoking single port LPM RAM

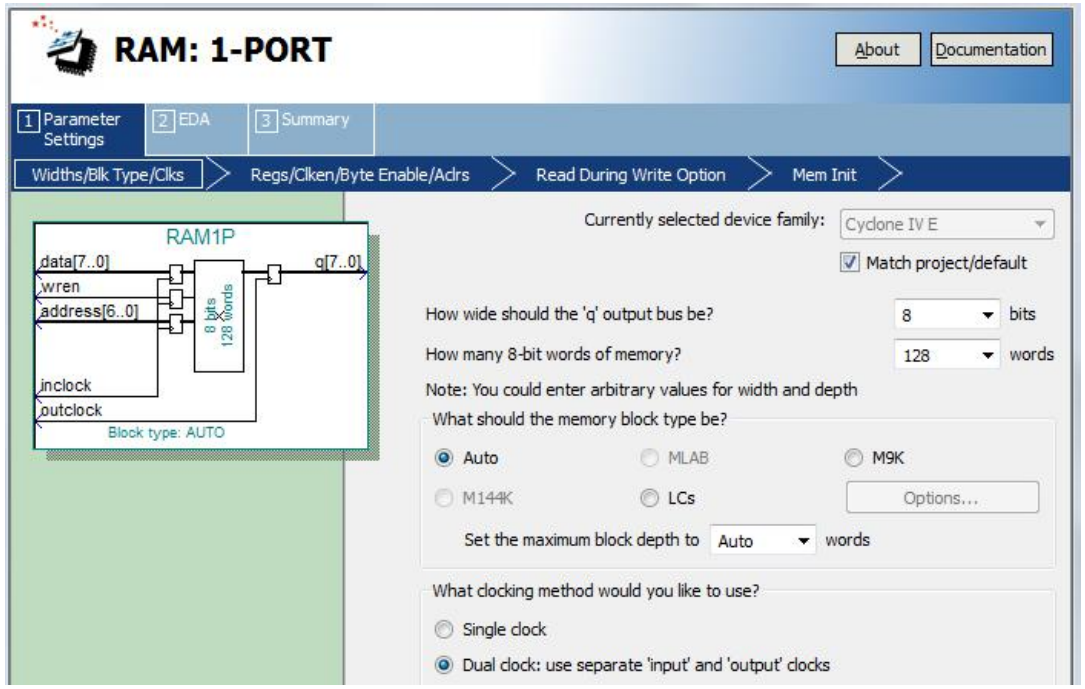


Figure: Setting RAM parameters

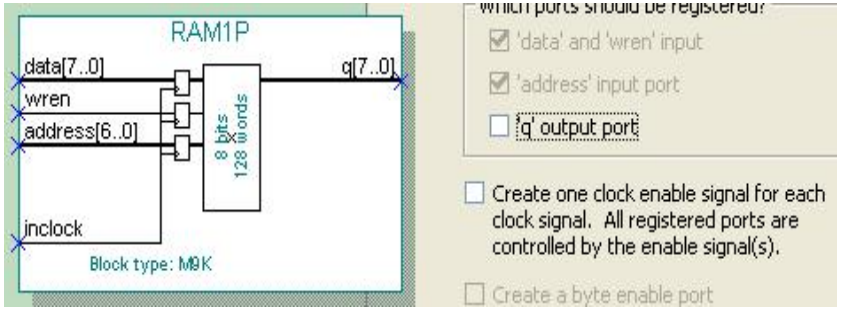


Figure: Setting RAM to be controlled by input clock only

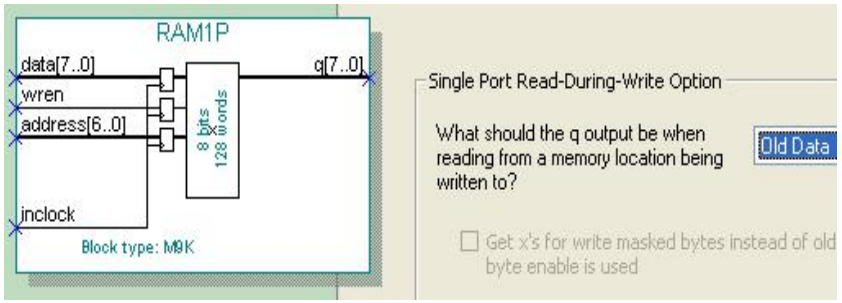


Figure: Setting to read the original data at the same time of writing data： Old Data

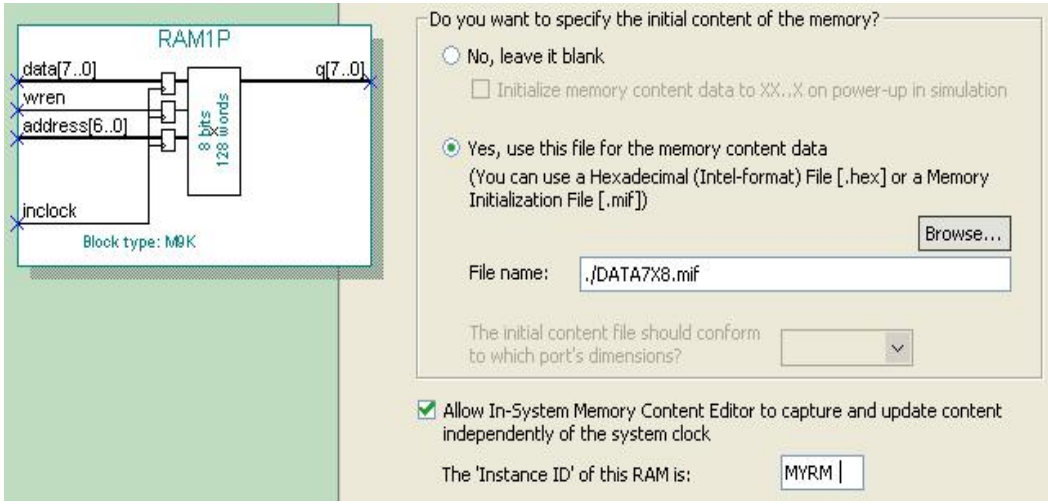


Figure: Setting the initialization file and allowing the in-system editing

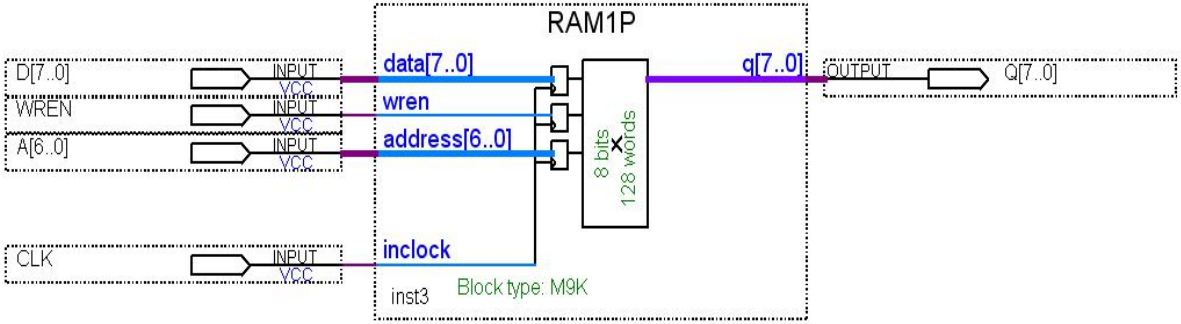


Figure: The well-connected RAM module on a schematic diagram

6.3.3 Test LPM_RAM

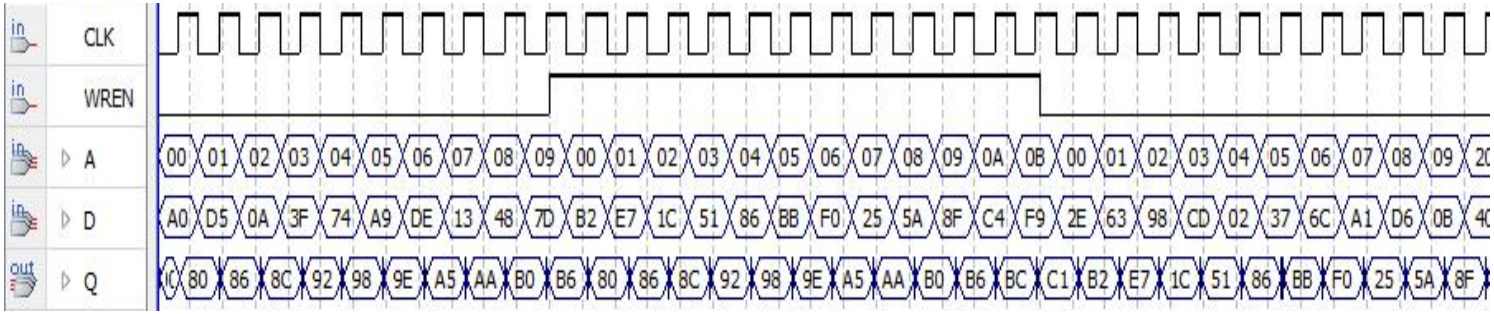


Figure: The simulation waveform of RAM

6.3.4 Expression of Memory Initialization File Loading of Verilog Code Description

- In Section 6.3.2, the reader has already seen that invoking the initialization file from the edited memory can use the editor called by the LPM module to select the settings in a specific dialog box (as shown in Figure 6-17). But if you invoke the initialization file in the memory of the Verilog program of the pure code, you must use a specific instruction statement. Here are two methods.
- The first method is to use the attribute statement given by Quartus II. These statements are used only in the Quartus II platform. On the right side of the memory definition statement of Example 2-3, there are:

- `/* synthesis ram_init_file="DATA7X8.mif" */ ;`

- The following definition expression is the Verilog-2001 version and its function is same:

- `(* ram_init_file = "DATA7X8.mif" *) reg[7:0] mem[127:0]`

The second method is to use the Verilog language directly, that is, using procedural statement *initial* and system function *$readmemh*. Because the standard Verilog statement is used, its expression has general characteristics, so it is not limited to EDA software environment of Quartus II. As “initial” and “$readmemh” are used, the format of initialization file must be “.dat”.

In .dat file, the data starts from lower address bit. Therefore, in example 6-6, the memory “mem” is written as mem[0:127].

[Example]

```
module RAM78 (output[7:0]Q, input[7:0]D,input[6:0]A, input CLK,WREN);
 reg[7:0]  mem[0:127] ;
 always @(posedge CLK ) if (WREN) mem[A] <= D;
  assign Q = mem[A];
  initial $readmemh("RAM78_DAT.dat", mem );
endmodule
```

6.3.5 Structure Control of Memory Design

The construction of memory in different Verilog expressions will obtain memories of different structures, such as a memory built by a logical macro unit or a memory built with an embedded RAM unit. The latter has the best resource utilization rate and the most concise and high-speed memory hardware structure for FPGA with large number of RAM units.

The RAMs described by Examples 6-6 and 6-7 have the same interface and function. Now let's compare their structure. The corresponding RTL diagram to Example 6-6 is shown in Figure 6-20; the corresponding RTL diagram to Example 6-7 is shown in Figure 6-21.

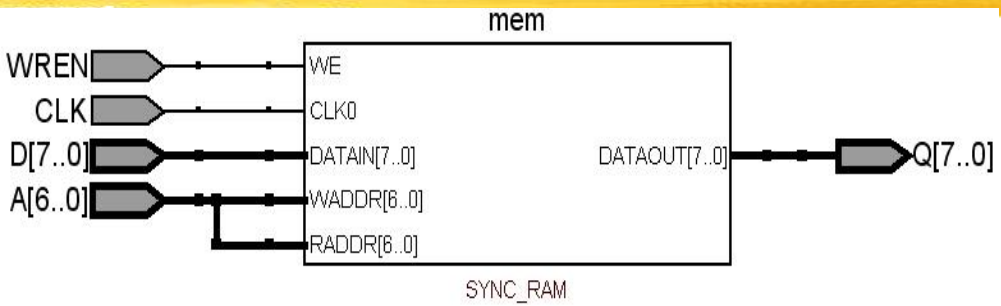


Figure: The RTL circuit module diagram of Example 6-6

[Example]

```
module RAM78(output reg[7:0] Q, input[7:0] D,input[6:0] A, input
CLK,WREN);
    reg[7:0]  mem[127:0]  /* synthesis ram_init_file="DATA7X8.mif" */;
    always @(posedge CLK )  if (WREN) mem[A] <= D;
    always @(posedge CLK )  Q = mem[A];
endmodule
```

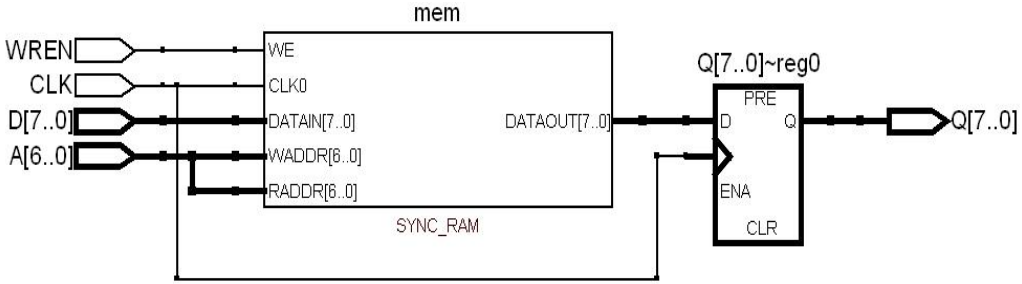
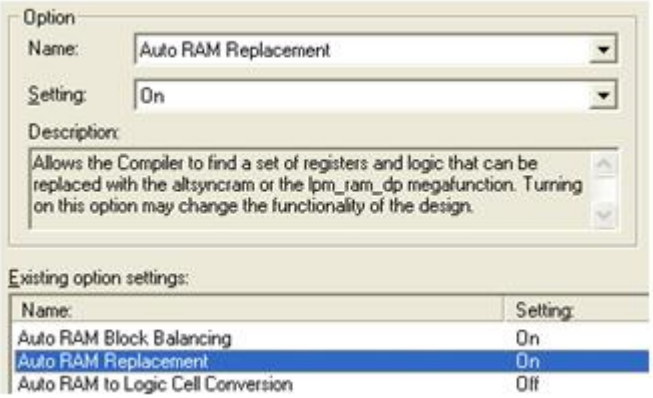


Figure: The RTL circuit module diagram of Example 6-7

- Why the differences between the example 7-6 and 7-8 are so big？The reasons are given as follows：
- （1）The expression way of Verilog. The output Q of the memory in Example 7-6 adopts “assign” statements. There is no any register or memory components in this case. Therefore, this structure can not use the ready-made RAM in the FPGA.
- Example 7-8 uses two “always” process and the output Q has an added register. This expression satisfies the RAM structure in the FPGA.

（2）invoke the constraint configuration of the embedded RAM units. The correct and proper Verilog descriptions is the basis of invoking the RAM units of the FPGA, which however can not guarantee that the design will invoke the RAM units. This is because the synthesizer still does not know the design purpose of the users. For constructing the circuit with the use of RAM after synthesis, the constraint configurations are needed for the synthesizer of EDA tools.



Settings->Analysis & Synthesis Settings->More Settings->Auto RAM Replacement->On

6.4 Usage Examples of LPM_ROM

6.4.1 Design of Simple Sinusoidal Signal Generator

MegaWizard Plug-In Manager -> Memory Compiler -> ROM:1-PORT

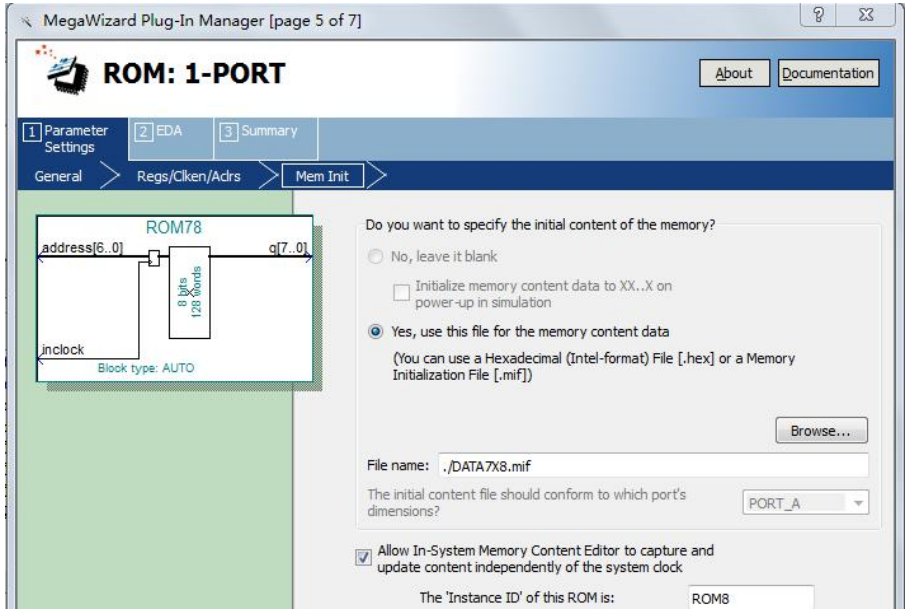


Figure: Adding the initialization configuration file and allowing in-system access to ROM content.

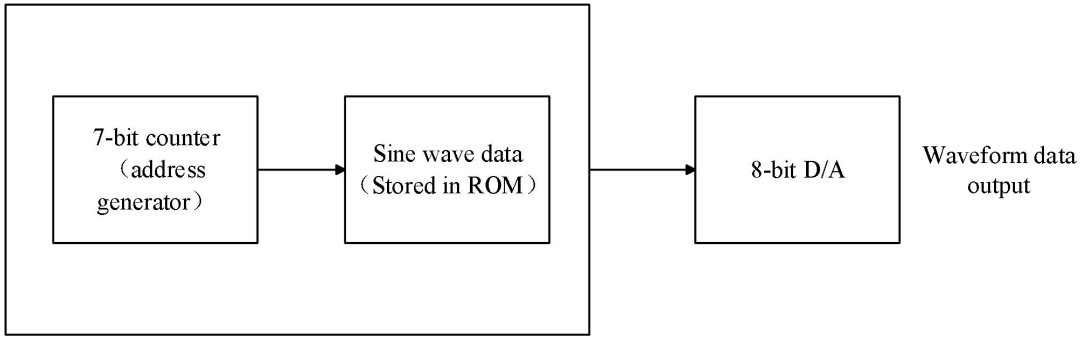


Figure: The block diagram of sinusoidal signal generator

- Counter or address signal generator. Here according to the parameters of ROM above, we select 7-bit output.
- Sinusoidal signal data memory ROM (7-bit address line, 8-bit data line), containing 128 8-bit waveform data (a sinusoidal period), that is, LPM_ROM: ROM78.
- Design of top-level schematic diagram.
- 8-bit D/A (set the experimental device to be DAC0832 for this example).

```
module SIN_GNT(RST,CLK,EN, Q,AR);
   output [7:0] Q ;  output [6:0] AR ;
   input  EN,CLK,RST ;   wire [6:0] TMP;    reg [6:0] Q1 ;
    always @(posedge CLK or negedge RST )
        if(!RST)   Q1 <=7'B0000000;
          else if (EN)   Q1 <= Q1+1 ;
          else  Q1 <= Q1;
    assign TMP=Q1;        assign  AR=TMP;
     ROM78  IC1(.address(TMP), .inclock(CLK), .q (Q) );
endmodule
```

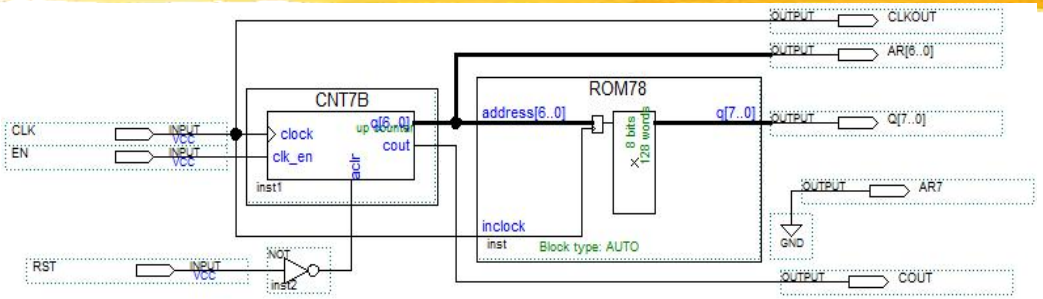


Figure: The circuit schematic diagram of sinusoidal signal generator

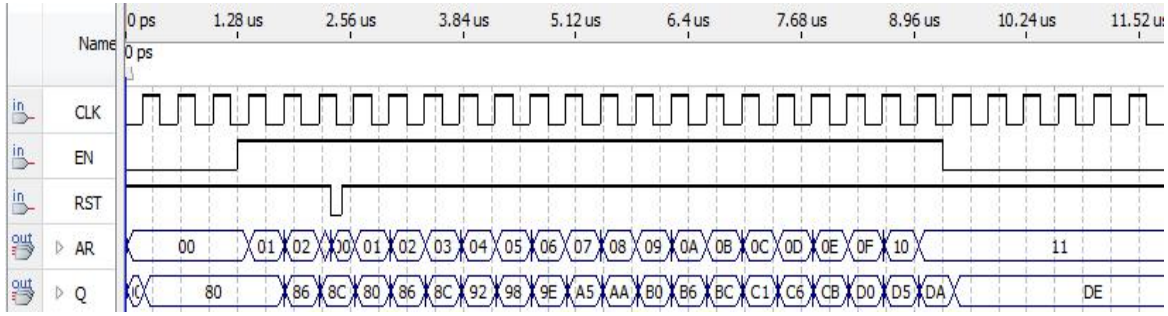


Figure: The circuit simulation waveform

6.4.2 Hardware Implementation and Testing of Sinusoidal Signal Generator

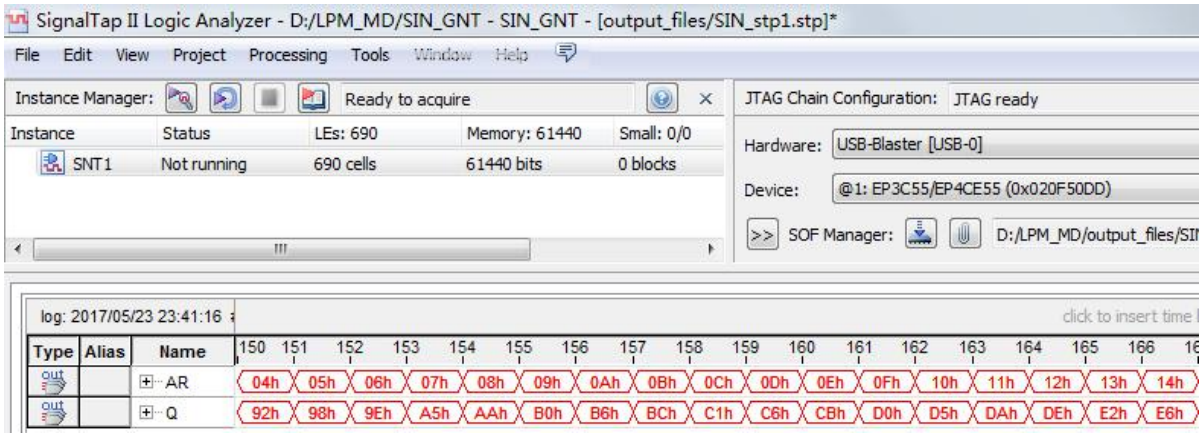


Figure: SignalTap II real-time test interface of sinusoidal signal generator data output

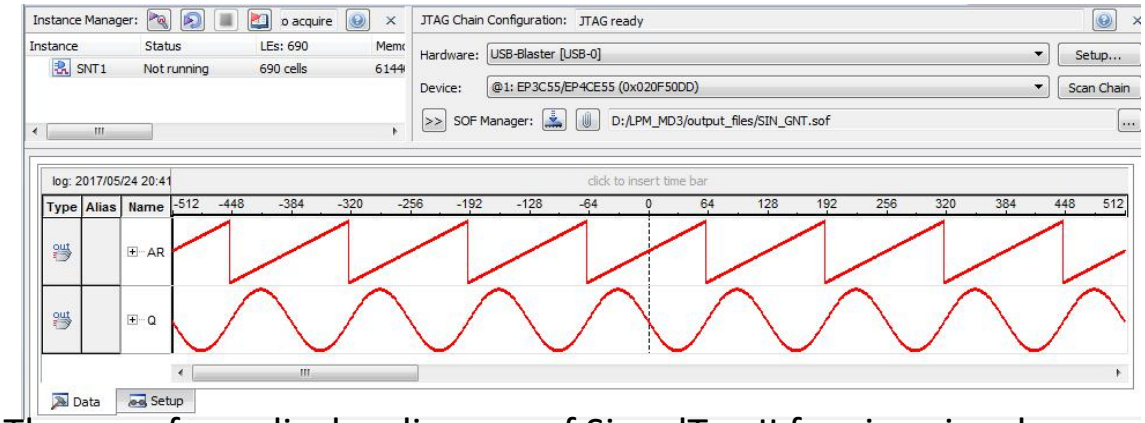


Figure: The waveform display diagram of SignalTap II for sine signal generator

6.5 Application of In-System Memory Content Editor

（1）Open the editing window of in-system memory content editor

The in-system memory content editor of Quartus II reads or writes the data of the operating memory of FPGA via JTAG port, and the read or write process does not affect the operation of the FPGA.

Tool->In-System Memory Content Editor.

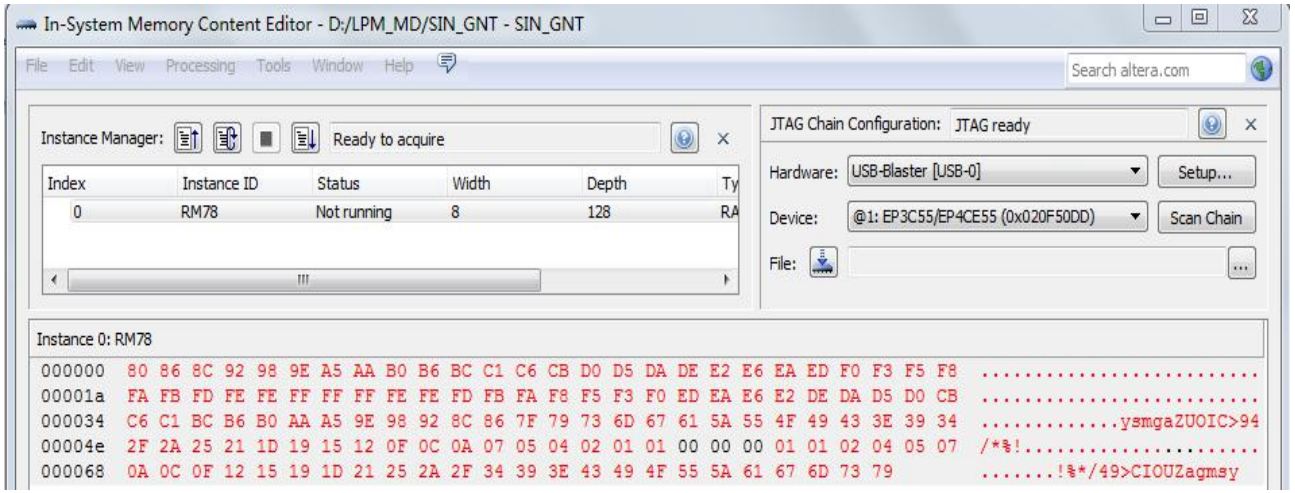


Figure: In-System Memory Content Editor editing window, reading waveform data from the ROM of FPGA

（2）Read the data of the ROM

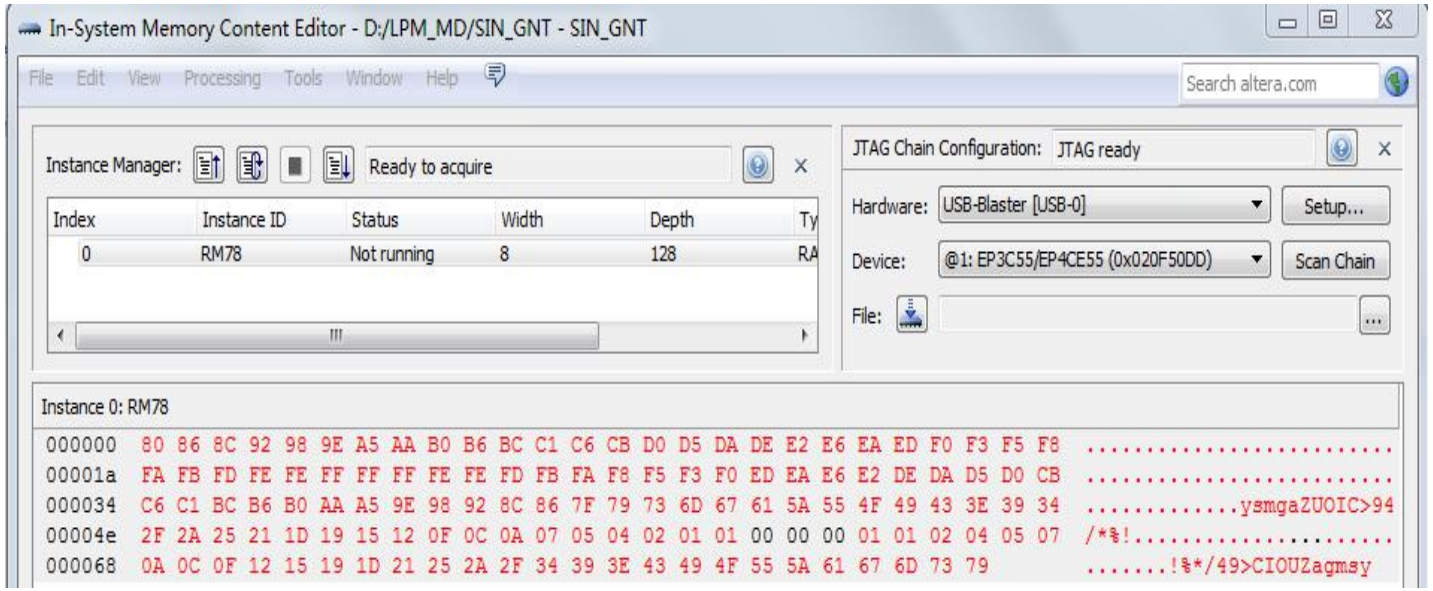


Figure: In-System Memory Content Editor editing window, reading waveform data from the ROM of FPGA

（3）Write data

Instance 0: RM78

000000	11	11	11	11	11	11	11	AA	B0	B6	BC	C1	C6	CB	D0	D5	DA	DE	E2	E6	EA	ED	F0	F3	F5	F8
00001a	FA	FB	FD	FE	FE	FF	FF	FF	FE	FE	FD	FB	FA	F8	F5	F3	F0	ED	EA	E6	E2	DE	DA	D5	D0	CB
000034	C6	C1	BC	B6	B0	AA	A5	9E	98	92	8C	86	7F	79	73	6D	67	61	5A	55	4F	49	43	3E	39	34
00004e	2F	2A	25	21	1D	19	15	12	0F	0C	0A	07	05	04	02	01	01	00	00	00	01	01	02	04	05	07
000068	0A	0C	0F	12	15	19	1D	21	25	2A	2F	34	39	3E	43	49	4F	55	5A	61	67	6D	73	79		

Figure: Here, the edited data is loaded into the ROM in the FPGA

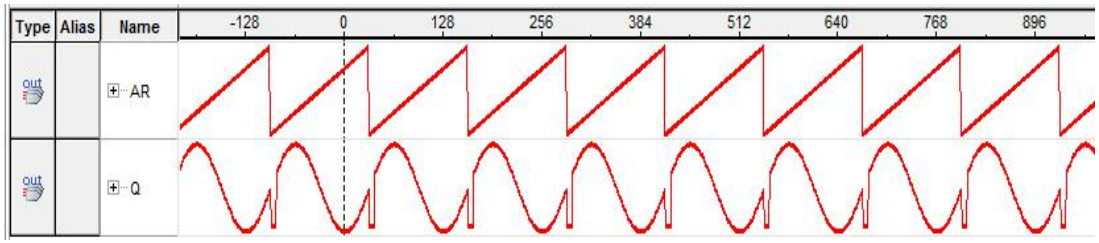


Figure: The data waveform measured by SignalTap II

After modification, choose Write Data to In-System Memory command in the processing menu. Then the edited data can be download to the LPM_ROM through JTAG port.

（4）The input and output data file

The data read from the system can be saved as MIF and HEX format file in the computer or “in-system” downloaded to the FPGA, through the command of Export Data to File or Import Data from File in the menu.

6.6 Invoke of Embedded PLL of LPM

The Cyclone/II/III/IV and Stratix/II/III/IV FPGA contain high performance PLL, which can be synchronized with the input clock signal. The input clock signal is also severed as the reference. The PLL can thus output one or several synchronized frequency scaling or frequency division on-chip clock.

6.6.1 Building Embedded PLL Component

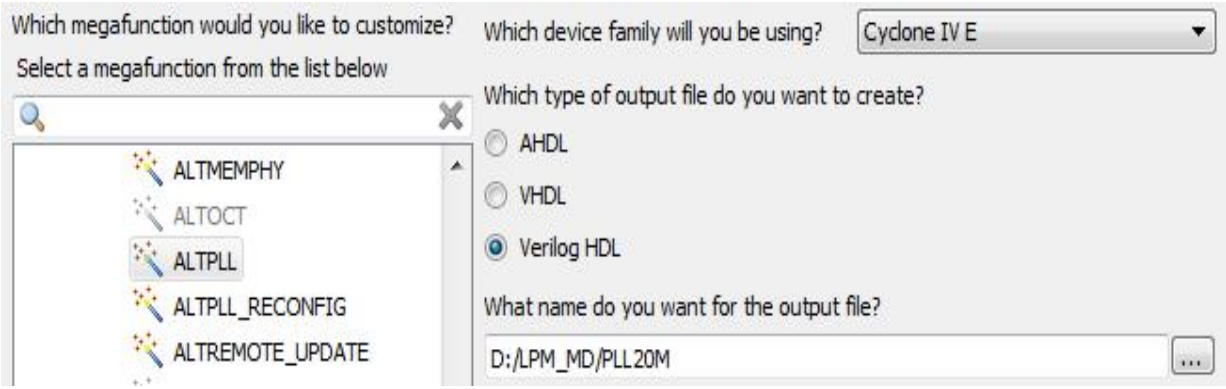


Figure: Choosing phase-locked loop ALTPLL

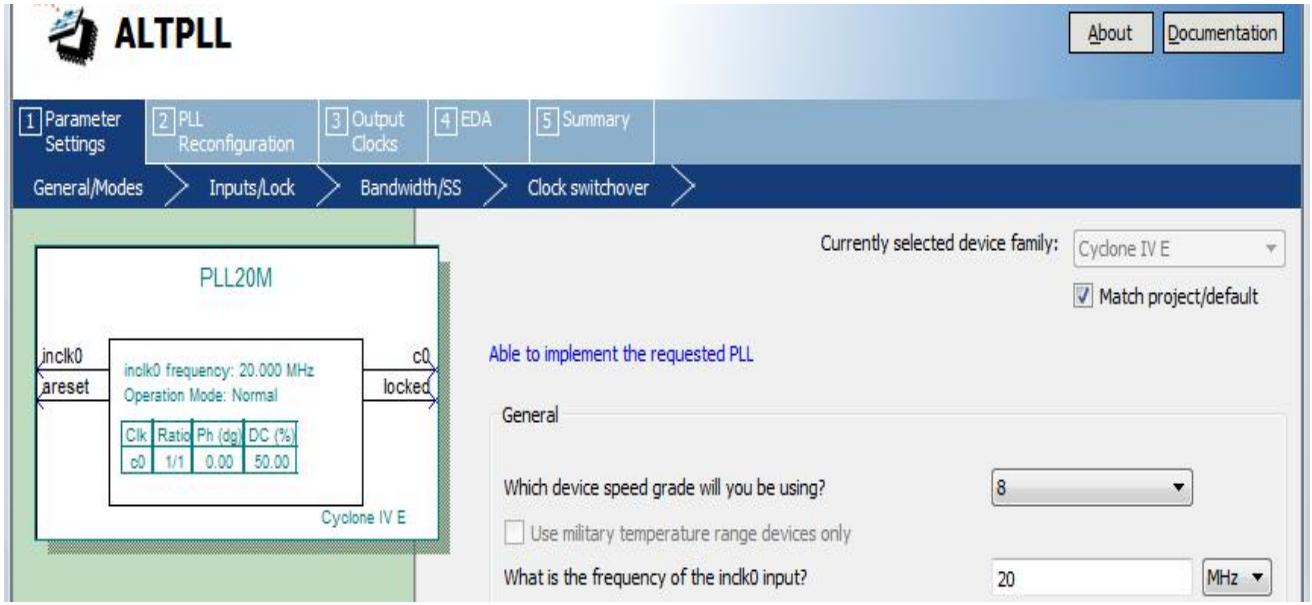


Figure: Selecting the input reference clock inclk0 to be 20MHz

MegaWizard Plug-In Manager->Create a new custom->I/O->ALTPLL

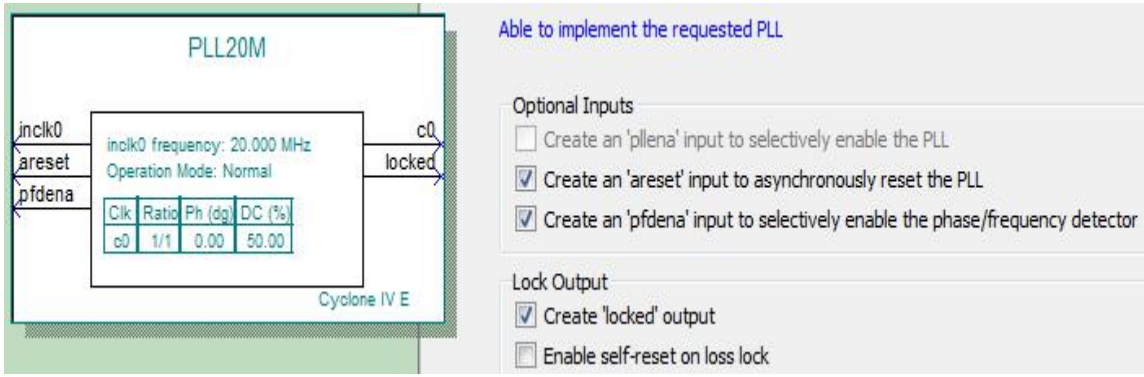


Figure: Selecting the control signal of the PLL

PLL enable: pllena
asynchronous reset: areset lock output: locked

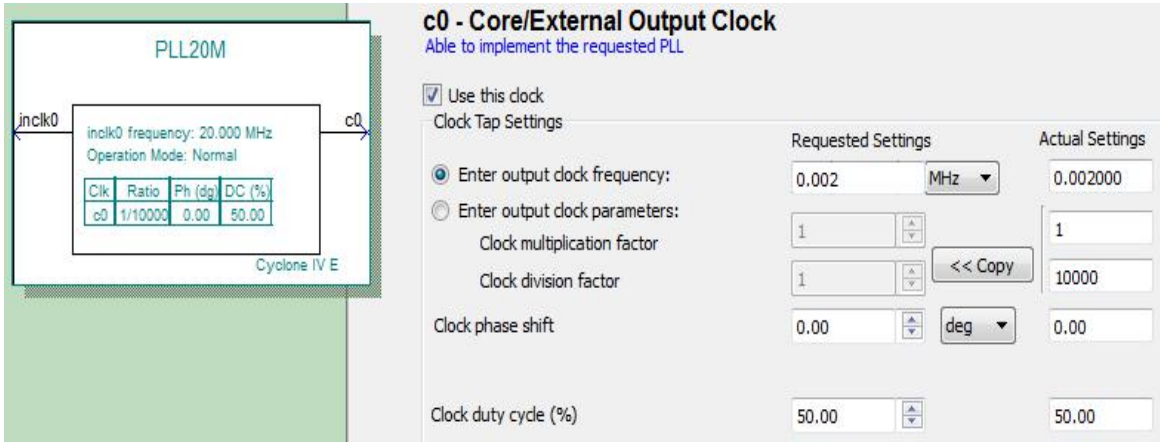


Figure: Selecting the output frequency of c0 to be 0.002MHz

Set up the frequency, phase and duty circle of the output clock

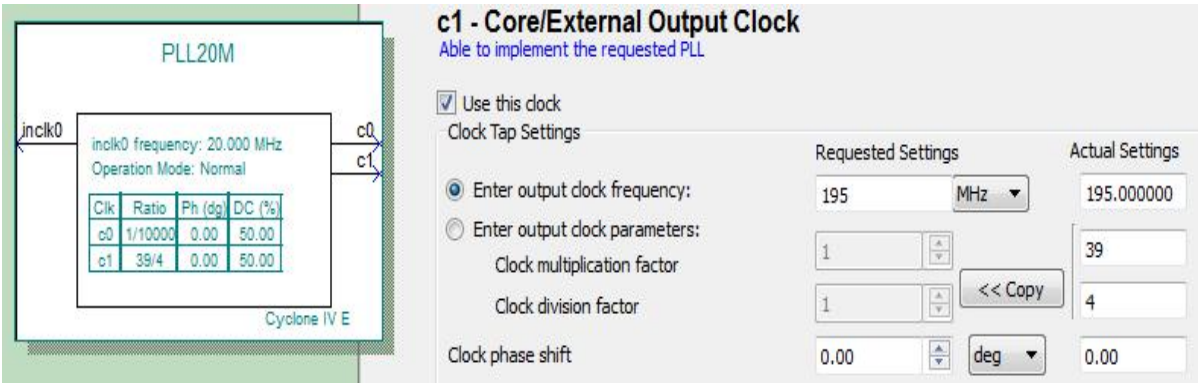


Figure: Outputing second clock signal c1

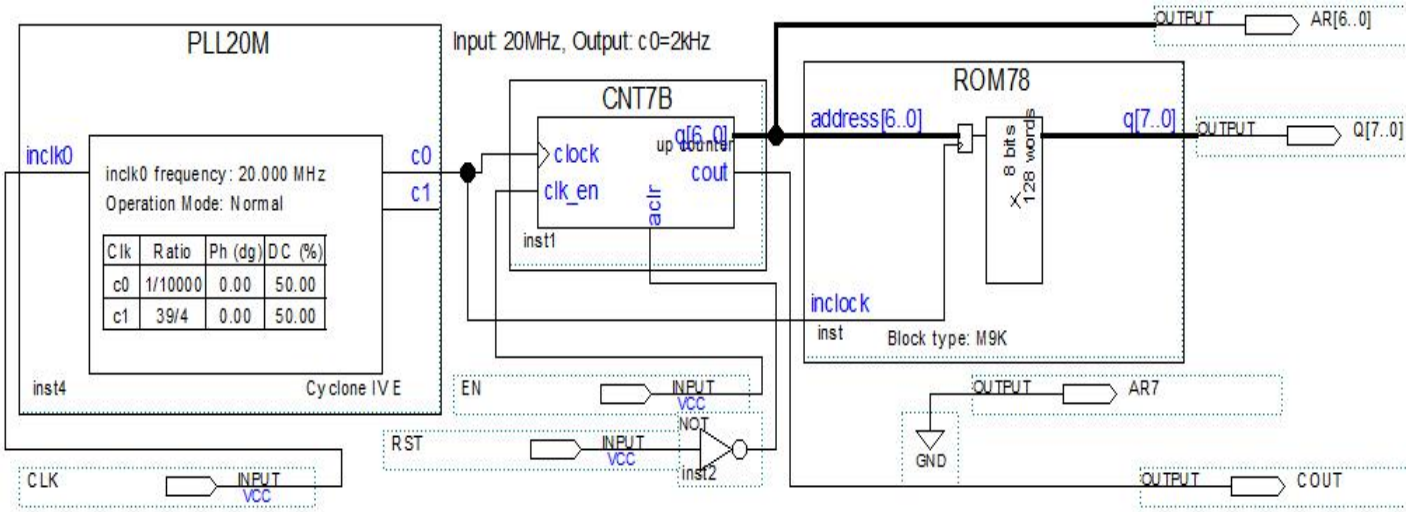


Figure: A sinusoidal signal generator circuit using embedded phase-locked loop as the clock

6.7 The Usage of In-System Sources and Probes Editor

SignalTap II and In-System Memory Content Editor can bring great convenience for logic system design, test and debug. However, they also have some disadvantages. For example, SignalTap II (1) occupies a large number of memory units as the data buffer; (2) unidirectionally gather and display the information of hardware system in the operation and can not interact with the system bidirectionally. In-System Memory Content Editor can interact with the system bidirectionally, the target of which is however only confined to memory.

In-System Sources and Probes Editor can overcome the above-mentioned drawback. In particular, the test signal for the system hardware does not have to connect to the I/O port (i.e. all the test signal internally connects to the test system) . All these tasks are realized by the communication through JTAG port of FPGA.

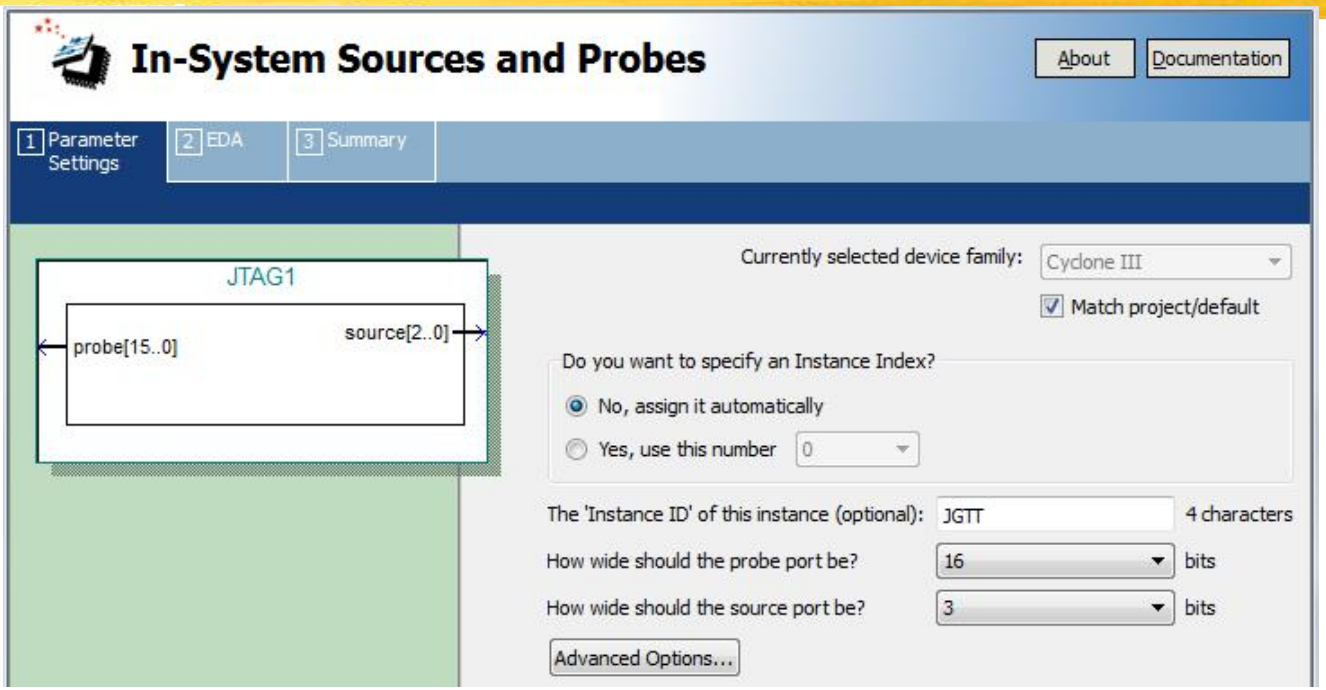


Figure: Setting the parameters for In-System Sources and Probes module

MegaWizard Plug-In Manager->Create a new custom megafunction variation->JTAG-accessible->In System Sources and Probes

The testing port "probe" of the "JTAG1" module is set up to be 16 bits and the signal source is 3 bits.

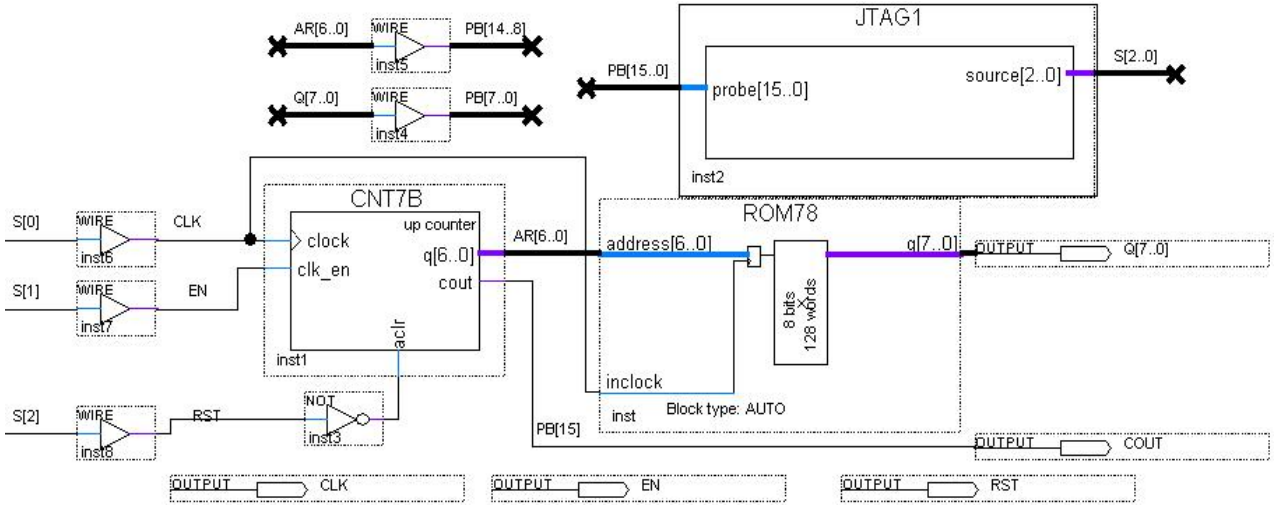


Figure: Adding In-System Sources and Probes testing module in the circuit

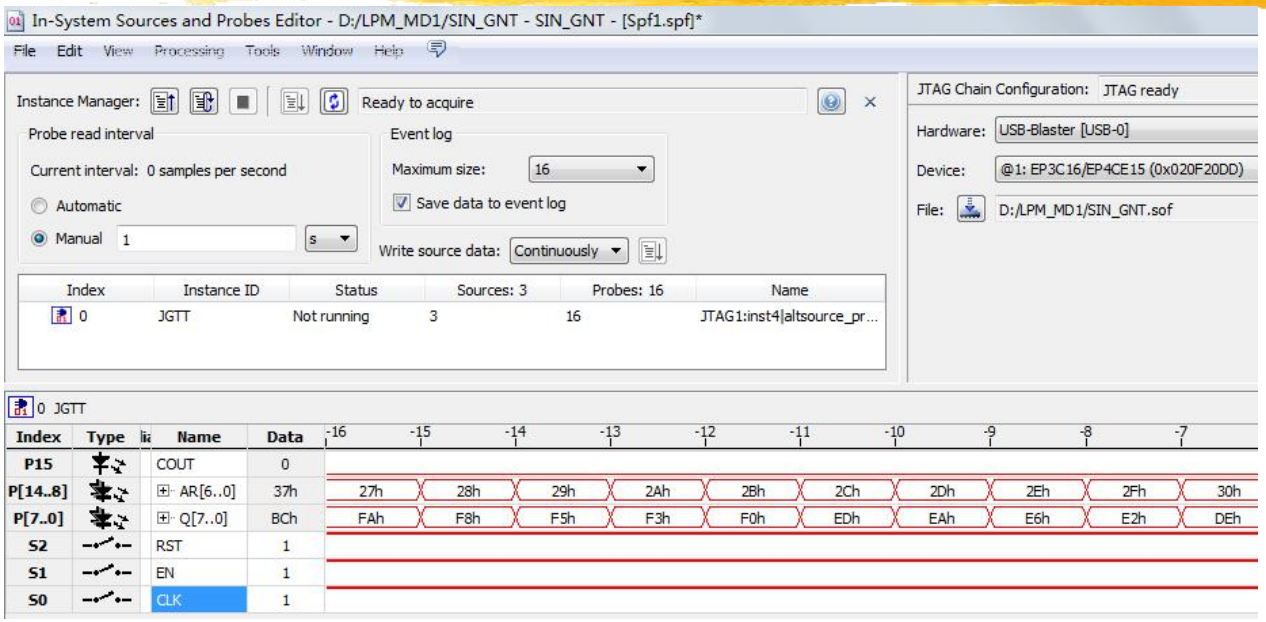


Figure: The testing condition of In-System Sources and Probes Editor

Tool->In-System Sources and Probes Editor

Maximum Size of Event Log refers to the number of sampling period, which is usually 8~32, (most of time is 32)。

S2、S1、S0 are the signals controlling the output of the source, which corresponds to RST、EN、CLK respectively.

6.8 Principle and Application of DDS

Direct Digital Synthesizer (DDS) is a frequency synthesis technology, which has high frequency resolution, can achieve fast frequency switching, and can keep the continuous phase in the change. It is easy to realize the numerical control modulation of frequency, phase and amplitude. Therefore, the application of direct digital frequency synthesizer is particularly extensive in the design of frequency source of modern electronic systems and equipment, especially in the field of communication. This section introduces the working principle of DDS and its hardware implementation.

6.8.1 Principle of DDS

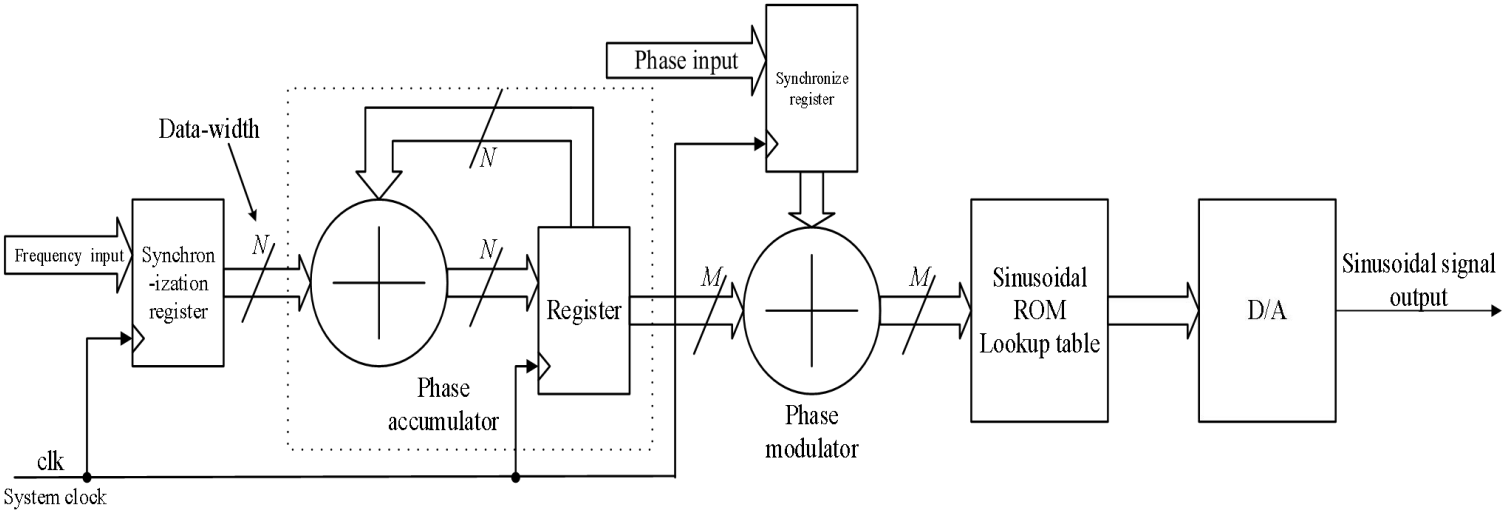


Figure: Basic DDS structure

DDS has the following four features.

(1) Theoretically, the frequency resolution of DDS can get the corresponding resolution accuracy when the bit number N of phase accumulator is large enough, which is difficult to achieve by traditional methods.

(2) DDS is an open loop system of fully digital structure without feedback link, so its speed is extremely fast, usually in the nanosecond order.

(3) The phase error of DDS mainly depends on the phase characteristics of the clock, and the phase error is small.

(4) The phase of DDS continuously changes, and the signal formed has good frequency spectrum, which cannot be realized by the traditional direct frequency synthesis method.

6.8.2 Example of DDS Signal Generator

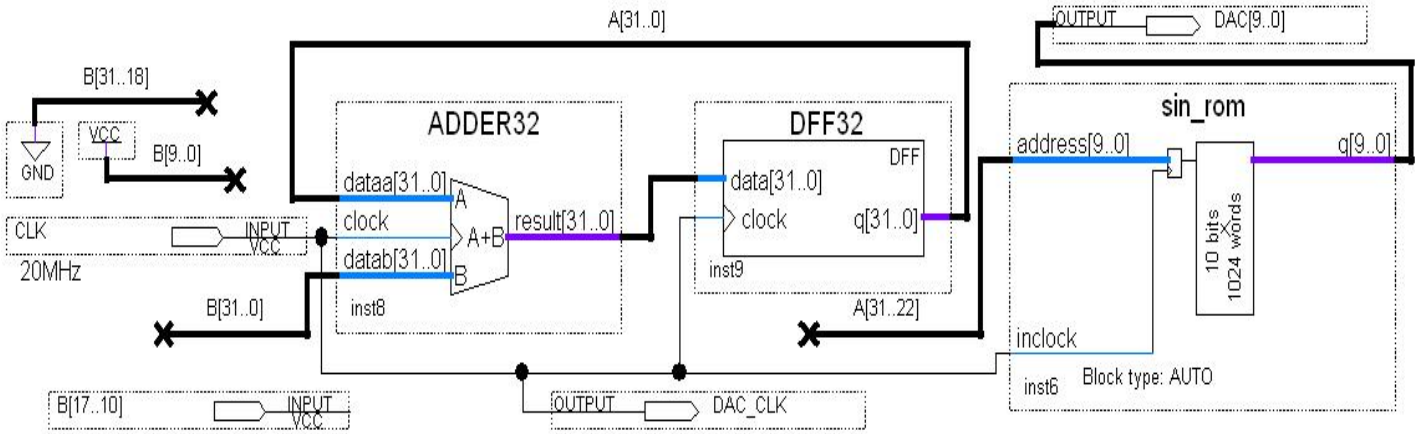


Figure: The top-level schematic diagram of DDS signal generator circuit

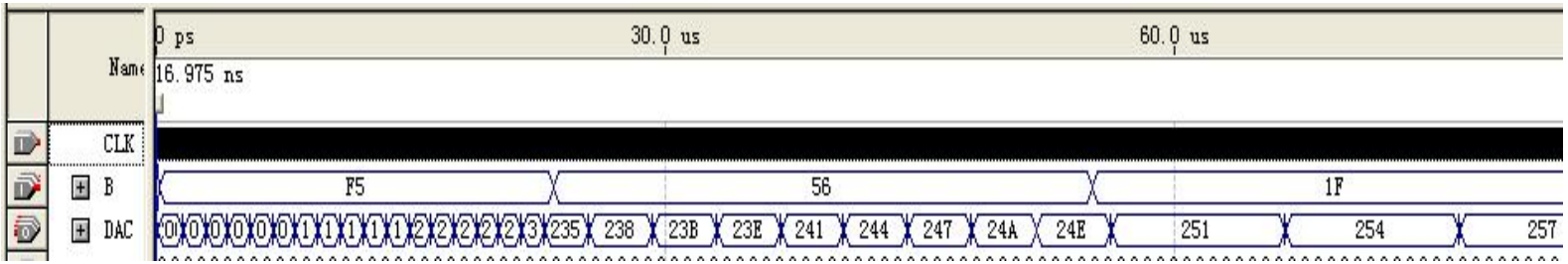


Figure: The simulation waveform of Figure 6-41